Introduction to SQL

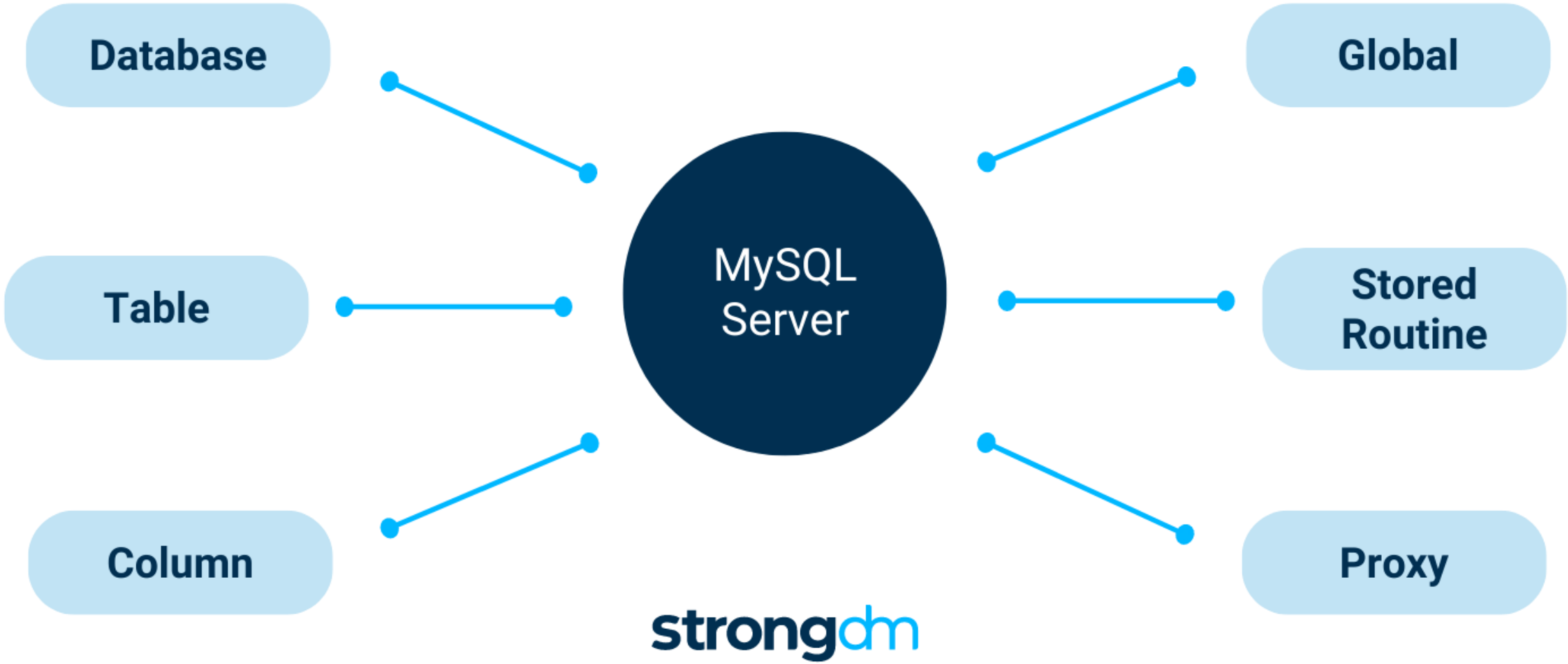
Class 8

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, FOR, LOOP), Stored Procedures,Stored functions
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

DCL – Data Control Language

MySQL Privilege Levels



strongdm

MySQL Privilege Levels

- **Global privileges** apply to all databases on the server. Administrative privileges fall into the global group because they enable a user to manage operations of the MySQL server and aren't specific to a particular database.
- **Database privileges** apply to specific databases in your MySQL instance and all of the objects within those databases (e.g. tables, columns, and views). You can also grant database privileges globally.
- **Proxy privileges** allow a user to act as if they have the privileges granted to another user.
- **Privileges for database objects** (tables, columns, stored routines, views, etc.) can apply to all objects of one type within a particular database or to specific objects, such as a certain table or view. You can also grant database object privileges globally.

DCL – Data Control Language

- Data Control Language (DCL) in MySQL is used to manage the access rights and permissions of database users.
- The two main DCL statements in MySQL are GRANT and REVOKE.
- GRANT statement is used to give specific privileges to a user or role on a specific database object (e.g. table, view, procedure, etc.).
- REVOKE statement is used to remove previously granted privileges from a user or role on a specific database object.
- MySQL supports both GRANT OPTION and REVOKE OPTION statements, which allow a user to grant or revoke privileges to other users.
- The privileges that can be granted or revoked include SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE, DROP, REFERENCES, INDEX, ALTER, and CREATE TEMPORARY TABLES.
- MySQL also supports the SET PASSWORD statement, which allows a user to change their own password or the password of another user.
- MySQL supports the CREATE USER statement, which allows to create a new user and assign different privileges to it.
- MySQL also supports the DROP USER statement, which allows to delete the user and all the privileges associated with it.
- MySQL also supports the RENAME USER statement, which allows to change the username of a user and keep all the privileges and grants associated with it.

Example commands

- SELECT user FROM mysql.user; shows all the users created and present
- SELECT current_user(); shows who the current user is.
- Create User Command
  - CREATE USER "local_user"@"localhost" IDENTIFIED BY "passowrd";
  - CREATE USER “local_user”@”localhost”;
- SET PASSWORD FOR 'jeffrey'@'localhost' = 'auth_string’; // For the given user
- SET PASSWORD =“password” // For current user
- DROP USER 'local_user'@'localhost’;
- SHOW GRANTS FOR ‘local_user‘@‘localhost‘;

Example commands - GRANT

- GRANT privilege ON privilege_level TO account_name;
- GRANT SELECT, INSERT ON strongdm.* TO ‘local_user’@’localhost’;
  - This statement gives SELECT and INSERT permissions on strongdm database to ‘local_user’@’localhost’
- GRANT ALL ON *.* TO 'janet'@'localhost' WITH GRANT OPTION;
  - This query provides all permissions to Janet whichever are present with the current user who is executing the statement
- GRANT SELECT ON database_name.table_name TO 'user_name'@'host_name';
  - This statement grants the SELECT privilege on the "table_name" table in the "database_name" database to the user "user_name" on the host "host_name".
- GRANT ALL PRIVILEGES ON database_name.* TO 'user_name'@'host_name';
  - This statement grants all privileges on all tables and views in the "database_name" database to the user "user_name" on the host "host_name".
- GRANT EXECUTE, CREATE ROUTINE ON . TO 'user_name'@'host_name';
  - This statement grants the EXECUTE and CREATE ROUTINE privileges on all databases and all stored routines to the user "user_name" on the host "host_name".

Example commands - REVOKE

- REVOKE SELECT, INSERT ON strongdm.* FROM ‘local_user’@’localhost’;
- REVOKE SELECT ON database_name.table_name FROM 'user_name'@'host_name';
  - This statement revokes the SELECT privilege on the "table_name" table in the "database_name" database from the user "user_name" on the host "host_name".
- REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'user_name'@'host_name';
  - This statement revokes all privileges and the grant option from the user "user_name" on the host "host_name"
- REVOKE DELETE, UPDATE ON database_name.* FROM 'user_name'@'host_name';
  - This statement revokes the DELETE and UPDATE privileges on all tables and views in the "database_name" database from the user "user_name" on the host "host_name".

ROLES – Some commands.

- CREATE ROLE 'role_name'@'host_name’;
  - Creates a role with the name role_name in the given host;
- GRANT SELECT, INSERT, UPDATE ON database_.table_name TO 'role_name'@'host_name’;
  - Gives the above permissions to the role
- GRANT 'role_name'@'host_name' TO 'user_name'@'host_name’;
  - Assigns the role to the user
- REVOKE 'role_name'@'host_name' FROM 'user_name'@'host_name’;
  - Revokes the role from a user
- SHOW GRANTS FOR 'user_name'@'host_name’;
  - Shows the roles assigned to a user
- DROP ROLE 'role_name'@'host_name’;
  - Deletes a role. If a role is deleted, then all the users who were assigned that role will not have that role anymore

MySQL Subqueries

- A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators.
- We can also nest the subquery with another subquery
- A subquery is known as the **inner query**
- The query that contains subquery is known as the **outer query**
- The inner query executed first gives the result to the outer query, and then the main/outer query will be performed.
- Subqueries should always use in **parentheses.**
- We cannot use the **ORDER BY** clause in a subquery, although it can be used inside the main query.
- If we use a subquery in a **set function**, it cannot be immediately enclosed in a set function.

MySQL Subqueries

- Advantages of using sub queries
  - The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
  - The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
  - The subqueries are more readable than complex join or union statements.

Subqueries examples

```
SELECT emp_name, city, income FROM employees
  WHERE emp_id IN (SELECT emp_id FROM employees);

SELECT * FROM employees
  WHERE emp_id IN (SELECT emp_id FROM employees
      WHERE income > 350000);

SELECT emp_name, city, income FROM employees
  WHERE income = (SELECT MAX(income) FROM employees);
```

Subqueries examples

```
SELECT Name, City FROM student
WHERE City NOT IN (
SELECT City FROM student2 WHERE City='Los Angeles');

SELECT Max(items), MIN(items), FLOOR(AVG(items))
FROM
   (SELECT order_id, COUNT(order_id) AS items FROM orders
   GROUP BY order_date) AS Student_order_detail;

SELECT emp_name, city, income
FROM employees emp WHERE income > (
SELECT AVG(income) FROM employees WHERE city = emp.city);
```

Subqueries examples

```
SELECT name, occupation, age FROM customer C
WHERE EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

```
SELECT name, occupation, age FROM customer C
WHERE NOT EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

```
SELECT cust_id, name FROM customer WHERE
cust_id > ANY (SELECT cust_id FROM Orders);
```

```
SELECT cust_id, name FROM customer WHERE
cust_id > ALL (SELECT cust_id FROM Orders);
```

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

Thank you